Problem 3-7: Scramble (Part 1)

This is CS50. Harvard University. Fall 2014.

Table of Contents

Questions? Feel free to head to CS50 on Reddit[1], CS50 on StackExchange[2], the `#cs50ap` channel on CS50x Slack[3] (after signing up), or the CS50 Facebook group[4].

Objectives

- Learn to read and build upon someone else's code.
- Learn how to encapsulate data.
- (Play.)

Recommended Reading

- Page 17 of http://www.howstuffworks.com/c.htm.

[1] https://www.reddit.com/r/cs50
[2] http://cs50.stackexchange.com
[3] https://cs50x.slack.com
[4] https://www.facebook.com/groups/cs50

- Chapters 20 and 23 of *Absolute Beginner's Guide to C*.
- Chapters 13, 15, and 18 of *Programming in C*.

Academic Honesty

This course's philosophy on academic honesty is best stated as "be reasonable." The course recognizes that interactions with classmates and others can facilitate mastery of the course's material. However, there remains a line between enlisting the help of another and submitting the work of another. This policy characterizes both sides of that line.

The essence of all work that you submit to this course must be your own. Collaboration on problems is not permitted (unless explicitly stated otherwise) except to the extent that you may ask classmates and others for help so long as that help does not reduce to another doing your work for you. Generally speaking, when asking for help, you may show your code or writing to others, but you may not view theirs, so long as you and they respect this policy's other constraints. Collaboration on quizzes and tests is not permitted at all. Collaboration on the final project is permitted to the extent prescribed by its specification.

Below are rules of thumb that (inexhaustively) characterize acts that the course considers reasonable and not reasonable. If in doubt as to whether some act is reasonable, do not commit it until you solicit and receive approval in writing from your instructor. If a violation of this policy is suspected and confirmed, your instructor reserves the right to impose local sanctions on top of any disciplinary outcome that may include an unsatisfactory or failing grade for work submitted or for the course itself.

Reasonable

- Communicating with classmates about problems in English (or some other spoken language).
- Discussing the course's material with others in order to understand it better.
- Helping a classmate identify a bug in his or her code, such as by viewing, compiling, or running his or her code, even on your own computer.
- Incorporating snippets of code that you find online or elsewhere into your own code, provided that those snippets are not themselves solutions to assigned problems and that you cite the snippets' origins.

- Reviewing past years' quizzes, tests, and solutions thereto.
- Sending or showing code that you've written to someone, possibly a classmate, so that he or she might help you identify and fix a bug.
- Sharing snippets of your own solutions to problems online so that others might help you identify and fix a bug or other issue.
- Turning to the web or elsewhere for instruction beyond the course's own, for references, and for solutions to technical difficulties, but not for outright solutions to problems or your own final project.
- Whiteboarding solutions to problems with others using diagrams or pseudocode but not actual code.
- Working with (and even paying) a tutor to help you with the course, provided the tutor does not do your work for you.

Not Reasonable

- Accessing a solution to some problem prior to (re-)submitting your own.
- Asking a classmate to see his or her solution to a problem before (re-)submitting your own.
- Decompiling, deobfuscating, or disassembling the staff's solutions to problems.
- Failing to cite (as with comments) the origins of code, writing, or techniques that you discover outside of the course's own lessons and integrate into your own work, even while respecting this policy's other constraints.
- Giving or showing to a classmate a solution to a problem when it is he or she, and not you, who is struggling to solve it.
- Looking at another individual's work during a quiz or test.
- Paying or offering to pay an individual for work that you may submit as (part of) your own.
- Providing or making available solutions to problems to individuals who might take this course in the future.
- Searching for, soliciting, or viewing a quiz's questions or answers prior to taking the quiz.

- Searching for or soliciting outright solutions to problems online or elsewhere.
- Splitting a problem's workload with another individual and combining your work (unless explicitly authorized by the problem itself).
- Submitting (after possibly modifying) the work of another individual beyond allowed snippets.
- Submitting the same or similar work to this course that you have submitted or will submit to another.
- Using resources during a quiz beyond those explicitly allowed in the quiz's instructions.
- Viewing another's solution to a problem and basing your own solution on it.

Assessment

Your work on this problem set will be evaluated along four axes primarily.

Scope
: To what extent does your code implement the features required by our specification?

Correctness
: To what extent is your code consistent with our specifications and free of bugs?

Design
: To what extent is your code written well (i.e., clearly, efficiently, elegantly, and/or logically)?

Style
: To what extent is your code readable (i.e., commented and indented with variables aptly named)?

To obtain a passing grade in this course, all students must ordinarily submit all assigned problems unless granted an exception in writing by the instructor.

Getting Started

First, log into cs50.io[5] and execute

[5] https://cs50.io/

```
update50
```

within a terminal window to make sure your workspace is up-to-date.

Then, execute

```
cd ~/workspace/unit3
```

at your prompt to ensure that you're inside of `unit3` (which is inside of `workspace` which is inside of your home directory). Then execute

```
mkdir scramble
```

to create a new `scramble` subdirectory inside of `unit3`. Navigate into that newly-created directory, and then:

```
wget http://cdn.cs50.net/ap/1516/problems/3/7/scramble.c
```

to download this problem's distro. Lastly, execute:

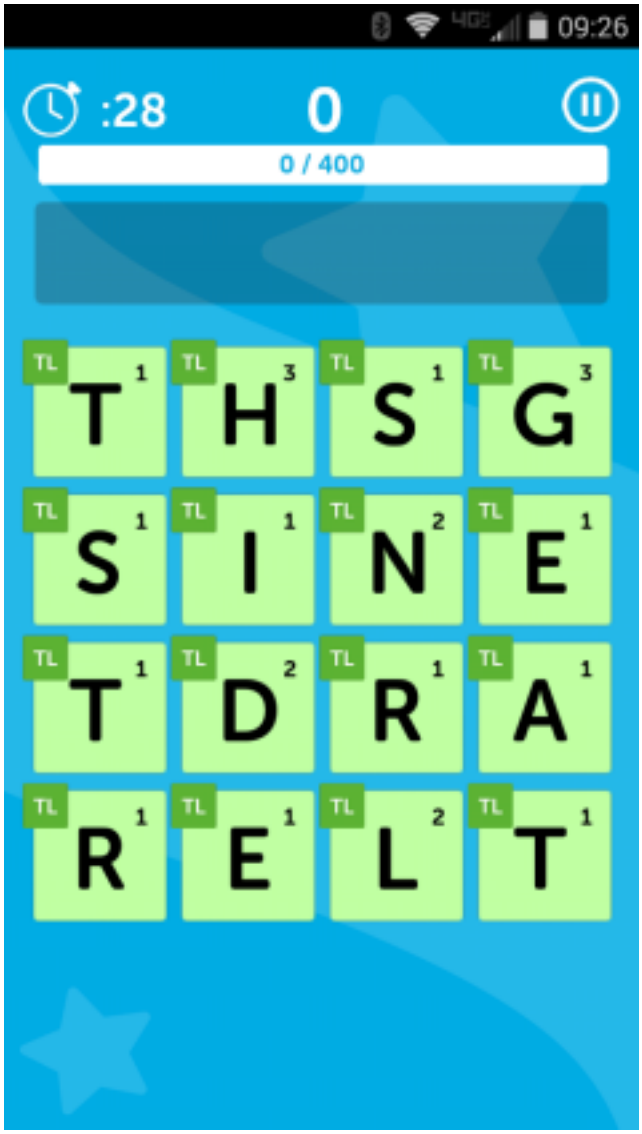
```
wget http://cdn.cs50.net/ap/1516/problems/3/7/words
```

to download a dictionary file with 172,806 English words. Hmm…

Scramblin'

Scramble was[6] a game, popular on smartphones, that challenges you to find as many words as possible in a 4x4 grid of letters before a timer expires. Each pair of letters in a word can be adjacent horizontally, vertically, or diagonally. Below, for instance, is what the modern version of the game looks like on an Android phone.

[6] The game is probably more commonly known—as of this writing—as "Word Streak with Friends".



Present are words like `THIS`, `SING`, `GEAR`, `HIS`, `RED`, along with many other words.

In your terminal prompt, execute:

```
~cs50/unit3/scramble
```

in order to try out the staff's implementation of `scramble`. You should see a 4x4 grid filled with letters. As soon as you spot a word, type it and hit Enter. If it's indeed a word in the grid (and in a dictionary of English words), your score will increase 1 point for each letter in the word. (Good job!) By default, you'll have 30 seconds to find as many words as you can. You won't see the clock ticking, but each time you input a word, you'll see how

many seconds you have left. As soon as time's run out, you'll be allowed to type one last word. (To quit the game early, hit ctrl-c.)

Now execute again:

```
~cs50/unit3/scramble
```

Odds are the grid of letters changed? That's because the distribution code uses `rand`. But what if you don't want the grid's letters to change each time you run `scramble`, particularly while debugging? No problem, simply execute

```
~cs50/unit3/scramble 1
```

to play grid #1, or

```
~cs50/unit3/scramble 2
```

to play grid #2, etc. That otherwise optional command-line arguments will be used as a "seed" for `rand` in order to perturb its output.

Perusin'

Time to take a break from playing! Open up `scramble.c`. The challenge at hand in this problem is to complete this game's implementation. But first, let's take a tour.

Notice first that atop `scramble.c` are a bunch of constants. Take note of the comments above each. Recall that declaring as constants values that you intend to use multiple times throughout your code tends to be good practice, so that you can change the value as needed in a single place.

Next, below those constants are some global variables. Global variables tend to be frowned upon (because there's usually a cleaner way to achieve some goal). But when the sole purpose of a file is to implement some program, as is the case here with `scramble.c`, it's not unreasonable to use globals to avoid passing around particularly important values again and again among several functions. For instance, we've declared `grid` as a global simply because so many functions will need access to it anyway, as you'll eventually see.

Notice next how we've utilized `typedef` and `struct` (both of which are quite new!) to declare our very own data type called `word`, inside of which is a `bool` and an array. We'll use a whole bunch of those structures in order to keep track of the words in that dictionary you downloaded (and whether the user has found them on the grid). Soon enough, we'll learn more about the `struct` keyword and how to create our own data types. For now, take for granted that it's possible, and consider it a sneak preview of things to come.

Below `word`, meanwhile, is `dictionary`, which we've declared as a `struct` without using `typedef`. The result is that this program will have just one `dictionary` structure, inside of which is an `int` and an array of many elements of type `word`.

Consider the prototypes below `dictionary` a sneak preview of the functions to come!

Incidentally, take care not to change any code related to `logfile`, which we use to automate some tests of your code!

Next read through `main`, focusing first on the comments and then on the code. If unsure at first glance what some line does, take some time to figure it out. It'll be a lot easier to write new code if you understand the code that's already there! If unfamiliar with some function, try to find it on CS50 Reference, else consult its "man page."

For instance, to pull up the manual for `atoi`, execute the below.[7]

```
man atoi
```

Notice, incidentally, how we're utilizing some "ANSI color codes" in main in order to output red text when the game's timer expires. They're a bit cryptic, to be sure, but pretty easy to use. See this link[8] for other colors.

Also, while debugging your program, you might want to comment out the call to `clear` in `main` so that you can see everything printed by `printf`, without anything disappearing.

Next read through each of the functions below `main`. Don't fret if you don't understand `find` and `crawl`, but do take a stab at reading through them. It turns out that `crawl`

[7] On occasion, you may need to execute `man 2 function` or `man 3 function`, where `function` is some function's name, lest you pull up the manual for a Linux command as opposed to a C function. For instance, the `man` page for C's `printf` is in (chapter) 3 and not 1, which is the default if you don't specify a chapter explicitly.

[8] http://pueblo.sourceforge.net/doc/manual/ansi_color_codes.html

implements a "recursive" algorithm (whereby `crawl` calls itself) that searches the grid horizontally, vertically, and diagonally for a particular word, "marking" letters temporarily as it visits them so that it doesn't accidentally get caught in an infinite loop.

Meanwhile, `initialize` might look a bit intimidating, but spend some time figuring out how it goes about initializing the grid with a distribution of letters. The `man` page for `rand` (albeit a bit cryptic itself) might help you figure out all the arithmetic.

Finally, `load` definitely has some new syntax, particularly `FILE`. We'll revisit `FILE` and more in the weeks to come. For now, know that `load` simply loads a whole bunch of words, one per line, from a file into an array.

Hm, it seems we forgot[9] to implement `draw`, `lookup`, and `scramble`. Oops!

Codin'

Suffice it to say we need your help finishing this implementation of `scramble`! And just a couple other favors, too, if you don't mind!

- Complete the implementation of `draw` (using some loops and `printf`) in such a way that `grid[i][j]` represents the letter in row `i`, column `j`. You're welcome to stray from the aesthetics of the staff's own solution.
- Complete the implementation of `lookup` in such a way that the function returns `true` if and only if `word` is in `dictionary`. Odds are you can do better than linear search!
- Complete the implementation of `scramble` (the function) in such a way that the grid is **rotated** 90 degrees (either clockwise or counterclockwise; that choice is up to you) anytime the user types `SCRAMBLE`, per the comments therein. Note that this feature doesn't affect the words in the grid; it simply lets the user see them from different perspective.
- By default, the distribution code is case-sensitive, whereby if `FOO` is in dictionary, the user must type `FOO`, not `foo`, in order to score. Alter `main` in such a way that the user can type `FOO` or `foo` (or even `FoO` or any other capitalization thereof) in order to score.

Now you can play (well, maybe after some debugging) your own version of `scramble`!

[9] Has the conceit of us "forgetting" to do certain things gotten old yet?

This was Problem 3-7.